下载APP

# 05 分析篇 | 如何判断问题是否由Page Cache产生的?

2020-08-29 邵亚方

Linux内核技术实战课 进入课程 >

讲述：邵亚方
时长 12:39 大小 11.60M

你好，我是邵亚方。

在前面几节课里，我们讲了 Page Cache 的一些基础知识，以及如何去处理 Page Cache 引发的一些问题。这节课我们来讲讲，如何判断问题是不是由 Page Cache 引起的。

我们知道，一个问题往往牵扯到操作系统的很多模块，比如说，当系统出现 load 飙高的问题时，可能是 Page Cache 引起的；也可能是锁冲突太厉害，物理资源（CPU、内存、磁盘 I/O、网络 I/O）有争抢导致的；也可能是内核特性设计缺陷导致的，等等。

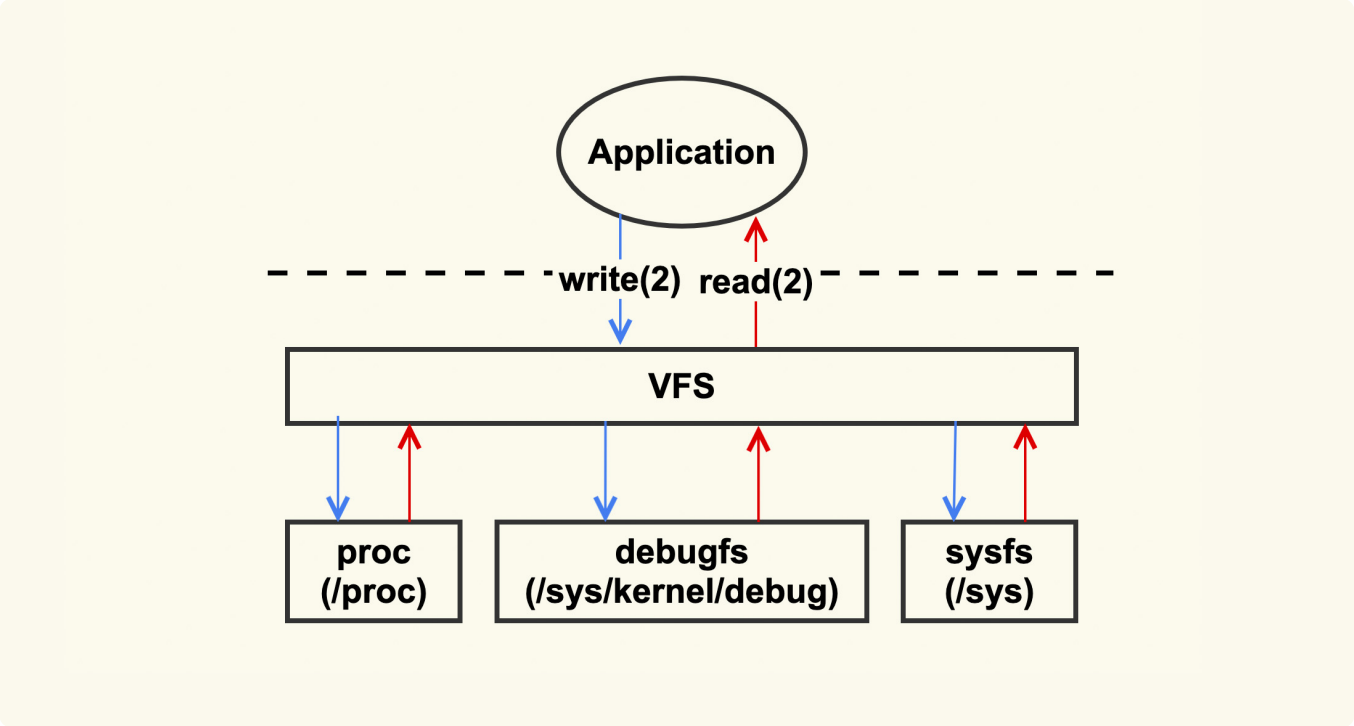
如果我们没有判断清楚问题是如何引起的而贸然采取措施，非但无法解决问题，反而会引起其他负面影响，比如说，load 飙高本来是 Page Cache 引起的，如果你没有查清楚原

因，而误以为是网络引起的，然后对网络进行限流，看起来把问题解决了，但是系统运行久了还是会出现 load 飙高，而且限流这种行为还降低了系统负载能力。

那么当问题发生时，我们如何判断它是不是由 Page Cache 引起的呢?

## Linux 问题的典型分析手段

Linux 上有一些典型的问题分析手段，从这些基本的分析方法入手，你可以一步步判断出问题根因。这些分析手段，可以简单地归纳为下图：



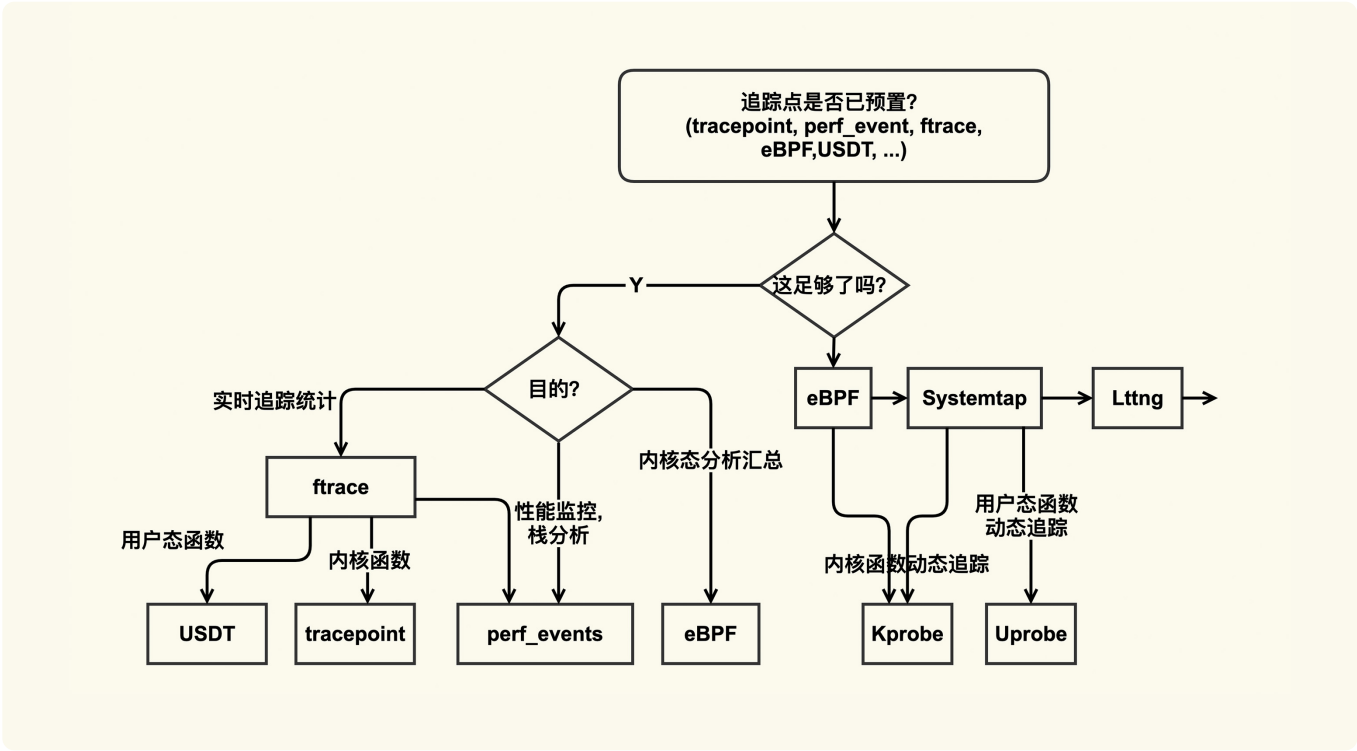

Linux的典型分析手段

从这张图中我们可以看到，Linux 内核主要是通过 /proc 和 /sys 把系统信息导出给用户，当你不清楚问题发生的原因时，你就可以试着去这几个目录下读取一下系统信息，看看哪些指标异常。比如当你不清楚问题是否由 Page Cache 引起时，你可以试着去把 /proc/vmstat 里面的信息给读取出来，看看哪些指标单位时间内变化较大。如果 pgscan 相关指标变化较大，那就可能是 Page Cache 引起的，因为 pgscan 代表了 Page Cache 的内存回收行为，它变化较大往往意味着系统内存压力很紧张。

/proc 和 /sys 里面的信息可以给我们指出一个问题分析的大致方向，我们可以判断出问题是不是由 Page Cache 引起的，但是如果想要深入地分析问题，知道 Page Cache 是如何引起问题的，我们还需要掌握更加专业的分析手段，专业的分析工具有 ftrace，ebpf，perf 等。

当然了，这些专业工具的学习成本也相对略高一些，但你不能觉得它难、成本高，就不学了，因为掌握了这些分析工具后，再遇到疑难杂症，你分析起来会更加得心应手。

为了让你在遇到问题时更加方便地找到合适的分析工具，我借用Bredan Gregg 的一张图，并根据自己的经验，把这张图略作了一些改进，帮助你学习该如何使用这些分析工具：




在这张图里，整体上追踪方式分为了静态追踪（预置了追踪点）和动态追踪（需要借助 probe）：

> 如果你想要追踪的东西已经有了预置的追踪点，那你直接使用这些预置追踪点就可以了；
>
> 如果没有预置追踪点，那你就要看看是否可以使用 probe(包括 kprobe 和 uprobe) 来实现。

因为分析工具自身也会对业务造成一些影响（Heisenbug），比如说使用 strace 会阻塞进程的运行，再比如使用 systemtap 也会有加载编译的开销等，**所以我们在使用这些工具之前也需要去详细了解下这些工具的副作用，以免引起意料之外的问题。**

比如我多年以前在使用 systemtap 的 guru（专家）模式的时候，因为没有考虑到 systemtap 进程异常退出后，可能不会卸载 systemtap 模块从而引发系统 panic 的问

题。

上面这些就是 Linux 问题的一些典型分析方法，了解了这些分析方法，你再遇到问题就能知道该选择什么样的工具来去分析。对于 Page Cache 而言，首先我们可以通过 /proc/vmstat 来做一个大致判断，然后再结合 Page Cache 的 tracepoint 来做更加深入的分析。

接下来我们一起分析两个具体问题。

## 系统现在 load 很高，是由 Page Cache 引起的吗?

我相信你肯定会遇到过这种场景：业务一直稳定运行着却忽然出现很大的性能抖动，或者系统一直稳定运行着却忽然出现较高的 load 值，那怎么去判断这个问题是不是由 Page Cache 引起的呢？在这里，我根据自己多年的经验，总结了一些分析的步骤。

分析问题的第一步，就是需要对系统的概括做一个了解，对于 Page Cahe 相关的问题，我推荐你**使用 sar 来采集 Page Cache 的概况**，它是系统默认配置好的工具，使用起来非常简单方便。

我在课程的第 1 讲也提到了对 sar 的一些使用：比如通过 sar -B 来分析分页信息 (Paging statistics)， 以及 sar -r 来分析内存使用情况统计 (Memory utilization statistics) 等。在这里，我特别推荐你使用 sar 里面记录的 PSI（Pressure-Stall Information）信息来查看 Page Cache 产生压力情况，尤其是给业务产生的压力，而这些压力最终都会体现在 load 上。不过该功能需要 4.20 以上的内核版本才支持，同时 sar 的版本也要更新到 12.3.3 版本以上。比如 PSI 中表示内存压力的如下输出：

复制代码

```
some avg10=45.49 avg60=10.23 avg300=5.41 total=76464318
full avg10=40.87 avg60=9.05 avg300=4.29 total=58141082
```

你需要重点关注 avg10 这一列，它表示最近 10s 内存的平均压力情况，如果它很大（比如大于 40）那 load 飙高大概率是由于内存压力，尤其是 Page Cache 的压力引起的。

明白了概况之后，我们还需要进一步查看究竟是 Page Cache 的什么行为引起的系统压力。

因为 sar 采集的只是一些常用的指标，它并没有覆盖 Page Cache 的所有行为，比如说内存规整（memory compaction）、业务 workingset 等这些容易引起 load 飙高的问题点。在我们想要分析更加具体的原因时，就需要去采集这些指标了。通常在 Page Cache 出问题时，这些指标中的一个或多个都会有异常，这里我给你列出一些常见指标：

| 问题点 | /proc/vmstat 指标 | 指标具体解释 |
| --- | --- | --- |
| Page Cache 回收 | pgscan_kswapd | kswapd 后台扫描的 Page 个数 |
| | pgsteal_kswapd | kswapd 后台回收的 Page 个数 |
| | pgscan_direct | 进程直接扫描的 Page 个数 |
| | pgsteal_direct | 进程直接回收的 Page 个数 |
| 碎片整理 | compact_stall | 直接碎片整理的次数 |
| | compact_fail | 直接碎片整理失败的次数 |
| | compact_success | 直接碎片整理成功的次数 |
| 脏页回写 | nr_dirty | 脏页个数 |
| drop cache | drop_pagecache | 执行 drop_cache 来 drop page cache 的次数 |
| | drop_slab | 执行 drop_cache 来 drop slab 的次数 |
| | pginodesteal | 直接回收 inode 过程中回收的 page cahe 个数 |
| | kswapd_inodesteal | kswapd 后台回收 inode 过程中回收的 page cahe 个数 |
| I/O | pgpgin | 从磁盘读文件读了多少 page 到内存 |
| | pgpgout | 内存中多少 page 被写回磁盘 |
| SWAP I/O | pswpin | 从 swap 分区读了多少 page 到内存 |
| | pswpout | 内存中多少 page 被交换到了 swap 分区 |
| workingset | workingset_refault | Page 被释放后短时间内再次从磁盘被读入到内存 |
| | workingset_restore | Page 被回收前又被检测为活跃 Page 从而避免被回收 |

采集完这些指标后，我们就可以分析 Page Cache 异常是由什么引起的了。比如说，当我们发现，单位时间内 compact_fail 变化很大时，那往往意味着系统内存碎片很严重，已经很难申请到连续物理内存了，这时你就需要去调整碎片指数或者手动触发内存规整，来减缓因为内存碎片引起的压力了。

我们在前面的步骤中采集的数据指标，可以帮助我们来定位到问题点究竟是什么，比如下面这些问题点。但是有的时候，我们还需要知道是什么东西在进行连续内存的申请，从而来做更加有针对性的调整，这就需要进行进一步的观察了。我们可以利用内核预置的相关 tracepoint 来做更加细致的分析。

| 问题点 | tracepoint | 解释 |
| --- | --- | --- |
| Page Cache回收 | mm_vmscan_direct_reclaim_begin,<br>mm_vmscan_direct_reclaim_end | 直接回收开始/结束 |
| 碎片整理 | mm_compaction_begin,<br>mm_compaction_end | 碎片整理开始/结束 |
| 脏页回写 | writeback_start,<br>writeback_written | 回写开始/结束 |
| 内存节点回收 | mm_vmscan_node_reclaim_begin,<br>mm_vmscan_node_reclaim_end | 内存节点回收开始/结束 |
| 磁盘I/O | block_rq_insert,<br>block_rq_issue,<br>block_bio_complete | 块请求插入/发出/结束 |

我们继续以内存规整 (memory compaction) 为例，来看下如何利用 tracepoint 来对它进行观察：

复制代码

```
#首先来使能compcation相关的一些tracepoing
$ echo 1 >
/sys/kernel/debug/tracing/events/compaction/mm_compaction_begin/enable
$ echo 1 >
/sys/kernel/debug/tracing/events/compaction/mm_compaction_end/enable

#然后来读取信息，当compaction事件触发后就会有信息输出
$ cat /sys/kernel/debug/tracing/trace_pipe
           <...>-49355 [037] .... 1578020.975159: mm_compaction_begin:
zone_start=0x2080000 migrate_pfn=0x2080000 free_pfn=0x3fe5800
zone_end=0x4080000, mode=async
           <...>-49355 [037] .N.. 1578020.992136: mm_compaction_end:
zone_start=0x2080000 migrate_pfn=0x208f420 free_pfn=0x3f4b720
zone_end=0x4080000, mode=async status=contended
```

从这个例子中的信息里，我们可以看到是 49355 这个进程触发了 compaction，begin 和 end 这两个 tracepoint 触发的时间戳相减，就可以得到 compaction 给业务带来的延迟，我们可以计算出这一次的延迟为 17ms。

很多时候由于采集的信息量太大，我们往往需要借助一些自动化分析的工具来分析，这样会很高效。比如我之前写过一个perf script来分析直接内存回收对业务造成的延迟。另外你也可以参考 Brendan Gregg 基于 bcc(eBPF) 写的direct reclaim snoop来观察进程因为 direct reclaim 而导致的延迟。

## 系统 load 值在昨天飙得很高，是由 Page Cache 引起的吗?

上面的问题是实时发生的，对实时问题来说，因为有现场信息可供采集，所以相对好分析一些。但是有时候，我们没有办法及时地去搜集现场信息，比如问题发生在深夜时，我们没有来得及去采集现场信息，这个时候就只能查看历史记录了。

我们可以根据 sar 的日志信息来判断当时发生了什么事情。我之前就遇到过类似的问题。

曾经有一个业务反馈说 RT 抖动得比较明显，让我帮他们分析一下抖动的原因，我把业务 RT 抖动的时间和 sar -B 里的 pgscand 不为 0 的时刻相比较后发现，二者在很多时候都是吻合的。于是，我推断业务抖动跟 Page Cache 回收存在一些关系，然后我让业务方调 vm.min_free_kbytes 来验证效果，业务方将该值从初始值 90112 调整为 4G 后效果立竿见影，就几乎没有抖动了。

在这里，我想再次强调一遍，调整 vm.min_free_kbytes 会存在一些风险，如果系统本身内存回收已经很紧张，再去调大它极有可能触发 OOM 甚至引起系统宕机。所以在调大的时候，一定要先做一些检查，看看此时是否可以调整。

当然了，如果你的 sysstat 版本较新并且内核版本较高，那你也可以观察 PSI 记录的日志信息是否跟业务抖动相吻合。根据 sar 的这些信息我们可以推断出故障是否跟 Page Cache 相关。

既然是通过 sar 的日志信息来评判，那么对日志信息的丰富度就有一定要求。你需要对常见的一些问题做一些归纳总结，然后把这些常见问题相关联的指标记录在日志中供事后分析，这样可以帮助你更加全面地分析问题，尤其是发生频率较高的一些问题。

比如，曾经我们的业务经常发生一些业务抖动，在通过我们上述的分析手段分析出来是compation 引起的问题后，而且这类问题较多，我们便把 /proc/vmstat 里 compaction 相关的指标（我们在上面的表格里有写到具体是哪些指标）记录到我们日志系统中。在业务再次出现抖动后，我们就可以根据日志信息来判断是否跟 compaction 相关了。

## 课堂回顾

好了，这节课我们就讲到这里，我们简单回顾一下。这节课我们讲了 Page Cache 问题的分析方法论，按照这个方法论我们几乎可以分析清楚 Page Cache 相关的所有问题，而且也能帮助我们了解业务的内存访问模式，从而帮助我们更好地对业务来做优化。

当然这套分析方法论不仅仅适用于 Page Cache 引发的问题，对于系统其他层面引起的问题同样也适用。让我们再次回顾一下这些要点：

> 在观察 Page Cache 的行为时，你可以先从最简单易用的分析工具比如 sar 入手，来得到一个概况，然后再使用更加专业一些的工具比如 tracepoint 去做更细致的分析。这样你就能分析清楚 Page Cache 的详细行为，以及它为什么会产生问题；
>
> 对于很多的偶发性的问题，往往需要采集很多的信息才能抓取出来问题现场，这种场景下最好使用 perf script 来写一些自动化分析的工具来提升效率；
>
> 如果你担心分析工具会对生产环境产生性能影响，你可以把信息采集下来之后进行离线分析，或者使用 ebpf 来进行自动过滤分析，请注意 ebpf 需要高版本内核的支持。

这是我沉淀下来的定位问题的方法。也希望你在遇到问题时不逃避，刨根问底寻找根本原因是什么，相信你一定也会有自己的问题分析方法论，然后在出现问题时能够快速高效地找到原因。

## 课后作业

假设现在内存紧张， 有很多进程都在进行直接内存回收，如何统计出来都是哪些进程在进行直接内存回收呢？欢迎在留言区分享你的看法。

感谢你的阅读，如果你认为这节课的内容有收获，也欢迎把它分享给你的朋友，我们下一讲见。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 04 案例篇 | 如何处理Page Cache容易回收引起的业务性能问题？

下一篇 06 基础篇 | 进程的哪些内存类型容易引起内存泄漏？

## 精选留言 (6)

写留言

**Jeff.Smile**
2020-08-29

sar 里面记录的 PSI（Pressure-Stall Information）具体怎么用啊？

作者回复: 这个是proc里面的文件，使用前提是，你的内核需要支持它，需要4.18以及更新的内核。如果内核支持了该特性，你就可以去/proc/pressure/里面去读取它了，这里面会有memory，io，cpu的信息。
你可以做一些工具来解析这些信息，具体做法可以参考sar的做法。
采集完这些数据后，你就可以使用它来作为系统压力指标的参考了。比如在业务有抖动时，你可以观察是否某个指标有异常。

4

**KennyQ**
2020-09-05

很多生产问题都是要对秒级甚至毫秒级的行为进行分析，而业务一旦向运维部门反馈了问题以后，一般都是要做事后分析， 那么一般如何应对这样的问题分析场景？
是针对一些重要指标在事前就进行秒级监控？分钟级监控？还是等待事后部署秒级的监控脚本进行信息抓取？

展开

作者回复: 如果采集的数据很多，那么秒级监控的开销还是很大的，所以一般都不会每秒去采集很多系统指标。通常情况下都是采用事件机制，比如在内核里一些关键路径上挂上钩子，当异常行为发生时就把该事件记录下来，但是这样做毕竟只是针对有限的事件，不会涉及到太多的事件，不然系统开销还是会大。所以在发生问题后的事后采集机制还是有必要的，因为发生问题后，运维或者业务会更加关注原因会是什么，对采集带来的开销会有心里预期，所以可以接受一定程度的损耗。很多时候借助这些粗力度的指标，是可以大致判断出问题可能出在哪里，然后再针对性的去做更细粒度的监控。
想要精确，就需要结合业务来深度定制监控；想要覆盖面广一些，就要尽量保障监控开销。鱼与熊掌是很难兼得的。

2

**我来也**
2020-08-30

邵老师，看了文中的一句话，正好有个困扰很久的疑问请教一下：
我们什么时候会真的遇到需要申请连续物理内存的情况？

> “单位时间内 compact_fail 变化很大时，那往往意味着系统内存碎片很严重，已经很难申请到连续物理内存了”...

展开

作者回复: 进程既有用户态也有内核态，在进程处于内核态时，就可能会申请连续内存。比如说进程要打开一个文件，那就会先查找该文件是否存在，查找的过程就是在内核态来完成的，然后这个过程中会分配文件所需要的一些内核结构体，比如dentry，inode等，这就需要申请内存，这些内存就是连续物理内存。

2

**ray**
2020-08-30

老师您好，请问
==> /proc/pressure/cpu <==

some avg10=0.00 avg60=0.00 avg300=0.00 total=10078249

==> /proc/pressure/io <==...

展开

作者回复: 1. 这个单位是应用程序阻塞时间，以memory为例，那就是进程在内存申请上消耗的时间。加入10s内，进程在内存申请上消耗了5s，那avg10就是50。

2.total代表进程总的消耗时间，包括历史累积值。以memory为例，那就是在内存申请上消耗的总时间。

这些在Documentation里都有说明

。

1 1

**jssfy**

2020-08-30

请问老师你们生产环境是否用4.18+内核多？还是定制迁移相关特性到自维护版本内核？

作者回复: 互联网企业普遍都是基于centos或者Ubuntu的内核，然后在这些内核的基础上来做自己定制化的特性，这些特性既有backport的 也有根据自己场景来实现的特定需求。生产环境上centos7是主流，内核为3.10，其次是centos8，内核为4.18。

1

**Eria**

2020-09-02

请教老师一个问题：我们两个机器上一样的系统和硬件配置和服务，运行相同的测试：

1. 系统 A 的磁盘 util 很小（3%-10%），但是可以看到 80G 左右的 buffer/cache，系统 A 的服务延迟非常小
2. 系统 B 的磁盘 util 很高（大于 30%），buffer/cache 10G 左右，系统 B 的服务延...

展开

作者回复: buffer/cache小的话，workingset refault会多，这会导致ioutil太高。为什么B中的buff/cache会比A中小那么多，我猜测是因为B中有很多不可以被回收的内存导致的，你可以观察下两个机器的/proc/meminfo,对比看看哪些项存在差异?